RESOURCE SHARING WITH SLIDING CONSTRAINTS

FIELD OF THE INVENTION

The present invention relates in general to mechanisms for guaranteed resource sharing, and more particularly to a system and method using sliding constraints for dynamically adjusting the priorities at which requests from applications in different request classes for a shared resource are processed.

BACKGROUND OF THE INVENTION

In many communications and computing systems, multiple applications share resources. Each of these applications requires access to other software and hardware resources of the communications and computing system. It is important that the sharing of these resources be done in a way that will fulfill the overall goals of the users and administrators of the system. Applications must have access to the resources of the system in a manner that is congruent with both the capacity of the system and the quality of services required by the application. In case of shortages of resources, more important applications should be given access to resources at the expense of less important applications so that more important functions can be maintained.

Typically, each application of a system is independent of others and the requirements of each application will vary from moment to moment. The sharing of resources of the system cannot be strategically fixed since the applications and the relative priority of these applications are all highly dynamic.

In a real time environment, it is necessary that real time applications get access to needed resources within the time constraints required of their tasks.

Real time systems can be characterized as those that must supply services that meet both absolute and relative time constraints. Real time systems are not necessarily those where the time requirements are short. A successful real time system is one that can meet the

time guarantees for the applications using them. The time requirements may be short or long. It is desirable that a real time system be able to set and adjust the timing commitments for the services that it can provide to applications and monitor compliance, taking remedial action if necessary. Real time operation is necessary for the performance of a service. It is also necessary to make run time decisions for allocations of resources.

In some cases, real time constraints are considered to be inflexible or hard. Hard real time constraints are system requirements whose violation would lead to the defeat of the system purpose and possibly severe system instability. It is desirable that a system be able to adjust priorities to ensure that hard time constraints are met.

There is always a specific set of resources available to any system. Of necessity, this set of resources is finite for open systems. Applications compete with each other for access to resources. Coordination and allocation mechanisms need to be put in place which partition these resources to applications in a way, which optimizes the performance to meet the overall system needs.

In the past, it has been common to use a mechanism for resource allocation that employs an economic model. Each application is allocated or initialized with a description of the necessary resources, including capacity and type of resource it requires, as well as an amount of economic utility which acts as a form of money with which to obtain or "buy" the resources. Applications enter a negotiation phase among themselves in which they buy and sell resources or access to resources to meet their needs. This economic model produces a rationality which can collapse the complicated logic of the relative importance of applications and allows for the sharing of resources in an attempt to minimize the expenditure of utility.

Applications using the economic model for resource allocation are structured to conserve the amount of utility that they expend. The mechanism is self-regulating. High priority applications are given more utility. They can use this in negotiation to outbid less important applications for the resources they need. In addition, applications can share resources and expend their utility across multiple resources. Applications must expend a higher amount of utility to obtain higher demand resources.

In a traditional communication or software system, the system must be designed to be reliable and manageable so that it will behave in a predictable manner and properly function. However, as systems become large and complex, they become difficult to understand, maintain and modify. As systems become more dynamic, understanding their temporal characteristics becomes more difficult, and therefore understanding and correlating the overall system behavior is very difficult and time consuming. Hence, in such complex systems, an agent approach has frequently been adapted to simplify the design and management of these complex communications and computing systems. An agent-based approach allows the functionality to be partitioned into a number of smaller, simpler components which are easier to develop, maintain, manage and supplement. Agent based solutions frequently provide a natural means of modeling a problem so that real world entities and their interactions can be mapped into autonomous problem solving agents which have their own resources and expertise and can interact with others to get the tasks done.

The concept of a general system using agents has been described in the publications "Toward A Taxonomy of Multi-Agent Systems", Int. J. Man-Machine Studies (1993), 39, 689-704, Academic Press Limited, and "An Intelligent Agent Framework for Enterprise Integration" by Jeff Y.C. Pan and Jay M. Tenenbaum, Transactions on Systems, Man and Cybernetics, (Vol. 21, No. 6, November/December, 1991, pages 1391-1407.)

Mihai Barbuceanu and Mark S. Fox, ("Integrating Communicative Action, Conversation and Decision Theory in a Coordination Language for Multi-Agent System", (1996) University of Toronto) have disclosed a language and design for providing objects and control structures to substantiate the construction of real multi-agent systems in industrial domains where agents communicate using structured conversations.

Tuomas Sandholm and Victor Lesser ("Issues in Automated Negotiations and Electronic Commerce; Extending the Contract Net Framework" in proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 328-335, San Francisco) have discussed issues that arise in automated negotiation among self interested

agents and have described a negotiation protocol and numerous message formats for negotiation of a contract between two agents.

The Spawn system (described in IEEE Transactions on Software Engineering, Vol. 18, No. 2, February, 1992, pages 103- 117), discloses a computational system which is organized as a market economy composed of interacting buyers and sellers. The spawn system allows processes in an operating system to purchase the CPU time they need.

Michael P. Wellman, in the paper entitled "A computational Market Model for Distributed Configuration Design", Proceedings of the Twelfth National Conference on Artificial Intelligence"; July 24, 1994 discloses a negotiation system that allows distributed systems to configure themselves out of a catalog of resources. However, none of the systems described above disclose an agent based negotiation system that is tailored for real time systems. In both the Spawn and Wellman systems, a purchaser buys the use of a resource. This results in the purchaser owning rights to a resource. In the case of real time systems, this strategy can have difficulties. A real time application, when it needs a resource, will typically not have sufficient time to negotiate for its service requirements. The level of service must be guaranteed beforehand.

Khasnabish B., "Integrated Bandwidth and Congestion Management in ATM Networks using a Simple Learning Algorithm and a Bucket Bank", 18th Conference on Local Computer Networks; Minneapolis Minnesota; IEEE Computer Society, 1993 pp. 388-396, describes a real time solution to resource management of a pool of bandwidth by multiple categories of ATM traffic on the basis of pre-assigned priority levels, discard eligibility, and current activity of the cells in a specific category. More particularly, Khasnabish describes a mechanism for learning priorities. The priorities are based on the length of the queue of each request class. This scheme is unsuitable to distinguish applications by their relative importance since the fact that an application makes many requests does not mean that it is important for the operation of the overall system.

There is a commercial need for a mechanism for negotiating a guarantee of service rather than the direct ownership shown in the previous models.

SUMMARY OF THE INVENTION

The resource sharing system according to the present invention dynamically adjusts the priorities at which requests from applications in different request classes (or classes of service) for a shared resource, are processed. The dynamic priority of a request class is based in part on the average resource allocation to requests in this request class, and in part on settings for the minimum and maximum allocations to the request class. The average resource allocation is the proportion of time the shared resource has been assigned to requests of this class relative to other classes.

The dynamic adjustment of priorities is referred to herein as the use of "sliding constraints" because the priorities are caused to "slide" with the average resource allocation, and the priority imposes a constraint on when the requests of a resource class can be satisfied: namely, when there are no other requests from higher-priority request classes waiting for the resource.

In contrast with Khasnabish, the system according to the present invention provides an economic mechanism for setting the parameters of priority functions that are a reflection of the relative importance of applications. For instance, if application A is more important than application B (see the call center example described in detail below), then application A should be given a larger portion of the resource than application B.

The sliding constraint mechanism of the present invention then uses these priority functions to dynamically allocate priorities to requests, the distinction being that the shape of the priority function is set by the economic mechanism to reflect the importance of the application, and not determined from observing the traffic generated by the application.

The shape of the priority functions determines the probability that a request will fail. If, for example, two priority functions share the same min parameter, but one has a higher max parameter than the other, the one with the higher parameter will fail less frequently,

since its requests will be satisfied more often. This represents a further improvement over Khasnabish.

BRIEF DESCRIPTION OF THE DRAWINGS

A more detailed description of the invention is provided herein below with reference to the accompanying drawings, in which:

Figure 1 is a declarative model of an architecture for the sharing of guaranteed resources;

Figure 2 is a relationship diagram of various agents used in the architecture of Figure 1;

Figure 3 is a block diagram of a blackboard system used for communication between agents in the architecture of Figure 1;

Figure 4 is a diagram illustrating the general structure of an agent of Figures 2 and 3;

Figure 5 comprises Figures 5A and 5B, and illustrates a block diagram of an agent of Figure 4;

Figures 6, 7 and 8 respectively comprises Figures 6A and 6B, 7A and 7B, 8A and 8B, and illustrate the functions of various routines of agents of Figure 4;

Figure 9 is a block schematic diagram of a goal resolution mechanism of an agent;

Figures 10, 11 and 12 illustrate various processes of creating new agents;

Figure 13 illustrates a blackboard process of Figure 3;

Figure 14 is a diagram of the present system embodying the present invention depicting higher level agents sharing the services of lower level agents;

Figure 15 is a graphic representation showing operation of the sliding constraints mechanism of the present invention in sharing resources between two classes of requests; and

Figure 16 is a flowchart showing the steps in processing requests for resources using the sliding constraints method of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

According to the present invention, resources are allocated to entities on the basis of negotiated guarantees of availability within the context of an economic model. In any real world implementation, the supply of known resources is fixed, but changes as resources are depleted or new resources are added and frequently certain resources are scarce. In a real world system the amount of resources available at any particular time for which there is resource contention is identifiable, quantifiable and allocable, and may be preset or determined by a configuration, registration or polling exercise. This exercise may be performed all at once, or in a stepwise fashion, with the services being first defined, then instances of the services being established, and finally the responsibility for the service being allocated to the responsible holding entity. This may be performed in hardware, firmware, or software and can be dynamically adjusted, rebalanced and redistributed as entity requirements change.

In a similar manner, entities during initialization or during system operation are endowed with a requirement for a service or resources as conditions warrant as well as "utility" for bidding. An entity may be a physical device, a program or application or a higher level abstraction thereof embodied in physical devices or applications operating in an intelligent manner such as an agent. Applications or processes may be stored as software in memory operating under control of a processor. An entity that requires a service or a resource makes a prediction of the amount of resource that it needs in order to meet its

expected load. This prediction is based on the possibility of the failure of the resource to be available at the instant that the entity needs it. This possibility of failure can be traded off against the price that the entity has to pay for the resource. Thus, the charge to the entity for the resource is not specifically for the quantity of resource used alone, but for the possibility of having that resource unavailable or removed. It should be appreciated that a resource or resources can be embodied in or managed by one or more entities. Initialized services or resources are assigned to holding entities and reallocated through appropriate signals, signaling methods and protocols. Also, bids requested as messages using such signaling methods and protocols are connected among entities. The invention is not limited to any specific method protocol or message format; any suitable signaling method, protocol and message format may be used.

In the preferred embodiment, the mechanism of the present invention is utilized through the use of agents. While the invention is described with respect to agents, the invention is not limited to the agent context. The invention can easily be adapted to any resource allocation situation. The invention is best described by way of an example of an architecture in which it is implemented.

Before describing the sliding constraints mechanism according to the present invention, a description is provided of Applicants' prior mechanism for the sharing of guaranteed resources as set forth in UK Patent Application No. 9916208.

The architecture of Figure 1 produces a mapping of concepts from the highest level of abstraction to the lowest. The architecture can be described in terms of five levels of abstraction. At the highest level of abstraction, the enterprise 10 can be viewed as a system with the set of goals to be achieved. The overall goals of the system are divided into smaller goals and distributed among functional groups 12. Each of these functional groups 12 is responsible for a portion of the overall goals of the enterprise 10. The goals of a functional group 12 are defined as an action to be performed using a set of services. Each service consists of a linked group of tasks 14 operating under functional groups 12. A task 14 can be viewed as a complex resource which has been tailored to the idiosyncratic needs of function group 12 of the enterprise 10. Each task 14 draws upon an abstract resource 16 which

consists of multiple physical resources which can share a common Application Programming Interface (API). The abstract resource 16 draws upon one or more physical devices 18.

Physical devices 18 described herein include processes, applications, devices, memories or machines that make up a resource 16 that can accomplish a task 14 or part of a task 14, and are defined by their capabilities and capacities. Processes and applications described herein are comprised of computer software executed on a processor, which includes any required program and data storage apparatus, such as random access or disk memories. Physical devices 18 are owned, and have their capabilities distributed via ownership. A physical device 18 is limited; external physical devices 18 are not aware of how tasks 14 are accomplished outside of the physical device 18. A physical device 18 may contain other physical devices 18, processes and agents internally, but these internal elements are not visible externally. The invention is not limited to any particular physical device and can include personal computers, servers, printers, telephones, switches, networks, data storage equipment, data transmission equipment or virtually any electronic or intelligent or intelligently controlled equipment.

In one preferred embodiment, physical devices include telephone interface circuits, trunk interface circuits, telephones, telephone lines, telephone line interfaces and telephone switches for establishing or maintaining a voice or data communication.

The mechanism of the present invention is described as operating to allocate resources 16 of physical devices 18, but can be employed between higher and lower level agents through the various levels of abstraction of Figure 1.

In order to utilize the present invention an initialization exercise is performed manually, during set-up or dynamically by an initialization mechanism that defines the character of the entities, the relationship between the entities, the services, the initial allocation of services among holding entities, the supply and character of resources and protocol for communications and negotiation. This is further illustrated in the example using agents.

AGENTS

An agent is an identifiable entity, as will be described below in more detail, which can accept a goal or goals, and produce an outcome. That outcome may be another goal or a set of goals. An agent may be the external representation of a physical device 18. A requirement or goal at the enterprise 10 level is mapped to a function provided by one of the functional groups 12. These are then mapped to tasks 14 at task level which draw upon abstract resources 16 at resource level which utilize the physical devices 18 at device level. Each level of the architecture provides a service to the layer above which is utilized by the layer above to achieve its goals. Communication between the layers can be done by agents. The interfaces between the various layers of the architecture and between various components in the architecture can be provided by agents. Furthermore, each layer optionally provides multiple service levels to attain various goals. The services provided at each layer are offered at various levels of capacity and quality. An example of a system employing agents is described in U.S. patent no. 5,638,494 of Pinard et al.

An agent only functions for the goals of which it has knowledge. A method for accomplishment of each goal is associated with each goal. These methods may involve some degree of planning or management, within the agent.

An agent may directly represent a physical device 18, or work through intermediate agents or intermediate physical devices 18, but is associated with its intermediate physical devices 18 through an abstract resource 16.

An agent sees only the portion of the capabilities of a physical device 18 or of another agent that it is entitled to use. This is referred to as the representation of the physical device 18 or of the resource 16. A resource 16 may consist of the representations of several devices and means for selecting from them. The means for selecting from several physical devices 18 is referred to as brokering.

A goal is an input to an agent, and specifies a task 14 or a sub task which an agent is to perform. Each goal is associated with a method for accomplishing the goal, as described in more detail later in this specification.

The capability of endowing an agent with goals and resources 16 is referred to as jurisdiction. Thus a higher level agent may use a lower level agent over which it has jurisdiction, as part of its goal definition, and thus it may endow the lower level agent with capabilities. A higher level agent with jurisdiction can provide another agent with a services using a lower level agent as a resource 16.

The right to use parts of the capability and the capacity of a resource 16 or physical device 18 is referred to as ownership, and the granting of rights to a physical device 18 to be used may only be done through ownership by an agent with jurisdiction. Ownership can be shared and may be of several types: constant, statistical, deterministic, or as available.

Ownership may be devolved through a hierarchy. Devolved ownership carries constraints: a devolved owner may devolve any type of ownership it has and add constraints, but may not remove constraints placed on a physical device 18 or another agent by a higher level agent. Ownership is also devolved on different time scales. Some higher level agents will require almost permanent ownership of a physical device 18. This ownership can be devolved dynamically, such as when a group allocates the rights to a group member for a single transaction.

An allocated physical device 18 may police its submitted goals to be sure that no other agent is exceeding its ownership rights. However, policing may not be necessary if done informally by policy.

The technique used by a resource 16 to select among physical devices 18 which can be used to accomplish a goal, is the allocation mechanism. The allocation mechanism is designed locally for the purposes of obtaining the resource 16. Together with the device representations, the allocation mechanism constitutes a broker within the resource 16. The broker is designed to achieve a purpose local to the resource 16, e.g. lowest cost, quality,

reliability, redundancy, most suitable device, etc. and participates in the bidding, utilizing a mechanism for sharing of guaranteed resources as described in further detail below.

Figure 2 illustrates a logical view of agent to agent communication used in the present system embodying the present invention. Jurisdiction is shown by a solid line arrow and usage rights is shown by a broken line arrow. An enterprise agent 20 has jurisdiction over all the other agents below it in hierarchy, e.g. group device agents 22, group user agents 24, etc. Similarly, the group user agent has jurisdiction over the user agents 26 below it, the group device agents have jurisdiction over the device agents 18 below them, etc. The group user agent 24 has usage rights over a portion of the service provided by group device agent 22. The user agent 26 has usage rights over a portion of services provided by the device controlled by the device agent under group device agent 28.

Brokers can select among resources the agent has usage rights over in order to better accomplish a goal, or can grant usage rights to another agent, or can set up goals and usage rights for its agent, or can customize resources which the agent has usage rights over.

A passive agent can only accept goals which can be accomplished without requiring resources from another agent. A passive agent is an atomic agent, typically representing a physical device 18.

Agents can be specialized for different types of work.

A group user agent 24 could represent a group of people which have been assigned a task to perform. A user agent 22 could represent a single person.

A group device agent 22 could be the initiator, or configurator, or creator of device agents 28 for a particular physical device type.

A device agent could represent data, or a task that a person can perform, or a physical device 18 including the actions of the device as well as setup information. This is a passive agent.

In the preferred embodiment, brokers and agents communicate using a blackboard system.

Blackboard systems have been described in the publications "Blackboard Systems", by Daniel Corkill, published in AI Expert, September 1991, pp. 41 - 47, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures" by H. Penny Nii, Published in The AI Magazine, Summer 1986, pp.38 - 53, and "Elevator Scheduling System Using Blackboard Architecture", by Grantham K.H.Pang, published in IEEE Proceedings-D, Vol. 138, No. 4, July 1991, pp.- 337 - 346.

As illustrated in Figure 3 a resource-requesting agent 30 requiring a resource posts a request to a blackboard (RAM) 32. This is interpreted by resource supplier agents 34 as a request for bids. Agents 34 then post bids to complete the process, in accordance with the resources 16 over which they have jurisdiction, and the economics of the completion of the job. In the present invention, the agents have particular design, and contain brokers, as will be described in more detail below.

The structure of an agent 30 is shown in general in Figure 4. The agent is categorized into four parts: an information area 52, a set area 54, an act area 56, and a resource area 58.

The information area 52 represents an area to which the agent posts information about its resources. Any agent which has usage rights over resources, or portions of resources, in this agent has read privileges for this area, if it can gain access to it. Access can be by direct read or be message based.

The set area 54 represents the ability of an agent to accept setup goals. In a passive agent, it can only accept goals which do not decompose into goals for other agents.

The act area 56 represents the ability of an agent to accept an acting goal and to decompose it into other goals which it passes on to other agents, or into resources to which it has usage rights. In a passive agent, goals can only decompose into resources that require no other agent interactions.

The resource area 58 represents the data and knowledge sources needed in the decomposition of a goal. It is private to the agent. A goal directory 60 breaks goals down into their constituent parts, is written to and from the set area 54, and is used from the act area 56. The representation of the resources including basic function, capacity, constraints, bidding mechanism, etc., is also contained in this area, as well as the knowledge source needed to utilize a resource. This can also be written from the set area 54, and used from the act area 50.

In order to set up a system of processes or applications, with agents that have no knowledge of other agents and what resources they use, or how they use them, an initialization procedure or an initializer mechanism is used. The initializer provides the initial setup of the system and defines the processes, decomposes them into goals for various agents, and the resources needed to accomplish each goal. One aspect of the initializer is that it characterizes the supply of the various services of holding entities available to bidding entities. In this manner, aspects of the system are defined. In a preferred implementation, this is accomplished by a system which defines in databases the enterprise 10 in terms of the organizational structure, including the users, the physical devices 18 and the resources 16 that they use. The processes that need to be done are described, including the users, groups and resources 16 needed to accomplish each part of them, and in what order. Once the databases are complete, they are decomposed into the goals and resources needed in each agent, and the agents are downloaded with this data. A process is defined as a series of goals, which need resources, and these goals are performed in a predetermined pattern. For example, a telephone call constitutes a process, a request to print data is a process, and an order to purchase equipment is a process of the enterprise 10.

Simple messages can be used to accomplish complicated tasks, since it is the interpretation of the messages by the agents which gives the system the ability to adapt and change to needs of the system. Services are created by process agents.

Thus services can be created dynamically by having a process agent, which has the sole task to create and maintain services which can have various levels of quality, reliability etc.

In this manner, the process agent acts as a provider of a supply of a service. In accordance with an embodiment of the system embodying the present invention, entities which request new services of processes post their request to an area of a blackboard. A request could for example come from an enterprise agent 20, as a dynamic request, or from an enterprise modeling tool which has collected via static input a process that needs to be added to the communication system. This spawns a process agent which is responsible for creating the requested process.

Figures 5 to 13 describe in detail one example of a signaling mechanism among agents for initialization, endowment, bidding, brokering and reallocation of services and resources among agents. It should be recognized that other signaling mechanisms could be used.

Figure 5 illustrates an agent 50 in accordance with the system embodying the present invention, and how it communicates with other agents. The solid arrows illustrate communication links during set-up, and the dashed line arrows illustrate communication links during operation.

The agent 50 is comprised of data in information area 52 and various routines 70, including the resource allocation mechanism 72. All of this is stored in memory. Routines shown in Figure 5 are capabilities definition mechanisms 70a, admission control mechanisms 70b, servant (task execution) mechanisms 70c, goal or plan resolution mechanisms 70d, remote customizing mechanisms 70e. Links are shown to external (other) agents 74, such as a supervisory agent, a subordinate agent, another agent that has usage rights over agent 50, another agent over which this agent has usage rights, etc. In this manner, as part of the initialization, or for dynamic adjustment of the system while in operation where agent 50 supplies services to a higher level agent or supervisor, agent 74 acts as an assignor that assigns a supply of services to agent 50 , with agent 50 responsible for holding and distributing its supply. In a similar manner, where agent 50 requires other services, a higher level or supervisor agent 74, (which may be another agent) acts as an endower that can endow agent 50 with utility for making requests for services.

The various routines 70a and 70b have functions as is further described with reference to Figures 6 and 7.

The capability definition mechanism 70a receives goals, tasks, policies and usage rights from an agent which has jurisdiction over this agent, i.e. from a higher level agent. It creates knowledge sources and updates the blackboard structure in the act area (56, Figure 4). The agent will also "know" how goals may be satisfied based upon how it is constructed and set up. The manner in which an agent can "know" is described by Daniel Dennet in her book "The International Stance". The ability of an agent to "know" how goals are satisfied facilitates the ability and directs the system to achieve those goals. This mechanism also places servant objects that can execute the possible tasks that will satisfy the different goals in the task executor 80. Representations are placed in a resource broker area 82 (representations of usage rights for resources in order to satisfy goals). This mechanism can add resources in excess of that provided by the group agent to meet local requirements; the local manager can "obtain" resources independent of the enterprise. It can also customize policies for the broker area.

Turning to Figure 7, the admission control mechanism 70b provides usage rights of this agent to other agents; it also provides performance guarantees to other agents regarding the resources in the resource allocation area 72. It contains methods for prediction of capacity and performance for the resources, including subsidiary agents, that it controls. The admission control mechanism 70b provides authorization and policing information to the act area 56, (in Figure 4). It contains a method to create lower level agents and to provide these agents with usage rights (goals and resources). The data contained in this area is comprised of the capacity and capability of the agent and the resources under control of the agent.

It is also through this area that another agent can provide notice that it is overwhelmed by the number and/or capacity of service requests or lacks resources, or another agent an trigger a request for bids, or notify that a resource is being taken away. The data in this area also includes a measure of the current amount of capacity and quality that is being used by other agents and the amount of unused capacity.

Turning to Figure 8, the resource allocation mechanism 72 contains a local representation of resources other agents that the agent has usage rights over, i.e. usage rights information, how much that agent is currently using of other agents or resources, and if necessary some information from the information or bulletin board area of the resource agent. This information is comprised of information relevant to the capability of other distant agent to provide the service that it has contracted from. This information from the distant information area, which can be in the form of resource representations can be updated periodically or instantaneously.

The agent takes part in a bidding process with another agent to supply resources to the agent, and exchanges usage rights guarantees and updates the resource provided a priori to the agent. When notice is provided by another agent for a request for bids, or that a resource is taken away, the notifying agent sends a message through the admission control mechanism 70b. The appropriate message is then passed to goal plan recognition mechanism 70d which sends message to task executor 80 which sends message to resource broker 82.

A resource broker 82 bids for resources for a task executor 80 of the agent based on policies which have been set up or previously endowed. Included within the endowment is the bidding resource allocation mechanism. The resources being bid for and obtained are stored as resource representations. The representation contains the address of physical resources, and thus the resource agent can update the representation for reasons of device failure or fault congestion, etc. The updating can be triggered dynamically, or periodically.

The resource broker 82 implements the bidding mechanism for the resource representations and send the resource allocation obtained back to the task executor 80, which posts feedback of the results to the bulletin board of the goal plan resolution area 70d. Appropriate notices are then sent back to supervising agents or other agents with usage rights over this agent.

With reference to Figure 9, as an alternative to the blackboard type of system, a goal resolution mechanism could be used where goals are stored in the information area 52 and listed in goal directories 90. Each agent has access to its own goal directory, which contains

a list of goals understood by the agent, and a procedure needed to accomplish each goal. Each also has access to its procedure table 92, which, for a given procedure, has a pointer to a software program 94 to run it. These software programs are run to resolve the goals of the agent, including allocation and use of resources.

Turning to Figure 10, in the case of an agent action to be created, it posts the request to blackboard 32. The process agent 96 that is chosen to construct the service creates a new procedure 98 which is downloaded into the agent 50. In this manner, a new service can be added and provided to other agents of the system. The service can also be updated and reconfigured in this manner. A new service can include the facilities to and access a new device of the system, as well as the features of the service available using the device. This also includes the features of quality and quantity of the resources available as part of the service.

Turning to Figure 11, in the case where a group device agent 32 wishes to create a new instance of a device agent 38, it can, through jurisdiction, download the new device agent 38 with the necessary software programs 100 to control a device controlled by the new device agent 38.

Turning to Figure 12, in the case in which a group user agent 34 has jurisdiction over a user agent 36, in can endow that user agent 36 with group specific goals 102 and procedures 104 necessary to accomplish them, for the operation of the group, over which the group agent 34 has jurisdiction.

Turning to Figure 13, when a significant event occurs in an agent 24, it posts that event to blackboard 22. Any agent 24a, 24b which is interested in the events registers that fact in registration area 110 of blackboard 22. When an event occurs, the blackboard is informed and all interested agents are notified, in a notification area 112.

The securing of a resource by a resource broker 82 is implemented through the concept of a holding price. The holding price is a measure of the amount, in terms of utility, that an agent must pay to hold a resource. An agent is endowed with utility to buy its needed

resources. The agent spends what it wants to get the quantity of resource that it needs. It must spend or pay the holding price continually to maintain and hold resources.

Turning to Figure 14, a generalized system embodying the present invention is illustrated with higher level agents 120, lower level agents 122 and a bidding mechanism 124. Each of the agents is endowed with the necessary goals, resources and procedures to implement the bidding mechanism. They are also endowed with their utility and priority. The initial appointment of utility may be done by any well-known means. For example, all entities could be given an equal amount of utility, then utility could be dynamically adjusted to reduce or eliminate system or device failure. Optionally, utility could be apportioned based on a ranking criteria in measures according to importance in avoiding failure of the system or a sub component of the system. In the context of agents, this is expressed as a failure to achieve goals. The ranking may be done by a system operator beforehand, or by some dynamic automatic mechanism. It is obvious to one skilled in the art that other means are possible. This could also be facilitated in a mechanism of sliding constraints as described in further detail below. Thus, the agent can spend a high amount for necessary resources and lower amounts for less important resources. An example of the bidding mechanism is illustrated below. The market system assumes that the resource has been fully allocated to other agents when an agent makes a bid for its needed resources. The original allocation may be facilitated during initialization and configuration of the system. In this example, the agent is not bidding for an unused resource but is bidding enough utility so that it can take resources away from existing agents currently using the resource. This illustrates how the system provides for dynamic changes in the supply of services, such as adding a new resource or capability, or removal or degradation of a service or resource, a change in the allocation of utility, or a change in the requirements for services or resources by an entity.

The mechanism set forth herein is based on the principle of taking resources away from agents that pay less than the bidding agent is willing to bid. This has numerous benefits. First, it makes any amount of resource available to an agent if it has enough importance. Second, it removes resources from less important agents allowing, the system to adapt to the overall needs of the enterprise. Third, it encourages efficient use of resources since an agent

must gain enough utility to pay for its resource. Therefore agents have an incentive not to squander utility on unneeded resources.

The following example describes a mechanism for performing this allocation, although, it is obvious to one skilled in the art that other mechanisms may be used.

An agent can be supplied with resources according to the following formulae.

Pc = current holding price of resource as held by an individual agent

Pb = bid price

Rc = current allocation of resource to agent

exp= exponent

Then allocation of resource to a bidding agent will be:

$Rb = Rc((1 - Pc/Pb)^{exp})$

if $Pc > Pb$ then $Rb = 0$

The exponent can be any value from 0 to 1 depending on the needs of the system. In the present example, the exponent of 0.5 is used for example purposes. This can be adapted so that different exponents can be used for the allocations of various resources in a system depending on the requirements of that environment.

The amount of Rb describes the amount of resource that will be taken away from an individual agent to supply a bid. Thus, prices for resources based on parameters such as Pc, Pb and Rc can be reasoned about concretely. The total supplied to the bidding agent will be the sum of the resources taken from all agents. This mechanism is used to generate an economic supply curve. Thus, prices can be established for any amount of resources from 0 to the maximum available.

The negotiation is conducted by a broker entity (such as an application or process), which in the agent context may be an agent or an aspect of an agent, using known communications protocols and messaging. Once the negotiation is complete, the broker communicates with the holding entities or agents for redistribution of the services or resources.

In the dynamic operation of the system, resources are sold and held by agents at a variety of amounts of utility. Agents paying high amounts of utility will not have their resources taken away from them by other agents. One clear example of resources being held by more important applications is the case of the trunk as a resource used for applications such as 911 calls and video conferences. The 911 call must never have resources taken from it without its permission. However a video conference is of less importance and can suffer poor quality without real harm to the enterprise. The mechanism above describes what proportion of an agents allocation will be taken away from it due to the bid of another agent. Thus if another agent bids more, a portion of its allocated resource will be taken away.

All agents who are paying less than a bidding agent is willing to pay will have a proportion of their resource allocation taken away. This means that an agent holding a resource can tell a bidding agent the amount that it will have to bid to get any quantity of resource up to the maximum. This will produce a supply curve for the resource which can be used by the bidding agent in its reasoning about the apportionment of its utility.

If a bidding agent is not willing to pay more for a resource than the current agent holding the resource, then it gets no resource from the holding agent.

Unallocated resources can be considered to be owned by the lower level agent. The holding price can be set to 0 or some higher amount to reflect the actual physical cost.

Returning to Figure 14, each of the higher level-agents 120 are sharing their services among other upper level services 126. The higher level agents 120 have a measure of their own capacity which they are supplying to the upper layers services 126. This capacity can be shared or multiplexed in any number of ways. It can also be done deterministically in which an upper layer service has exclusive use over a portion of the capacity. It can be done statistically in which a number of upper level services will share a portion of the capacity.

The amount of statistical sharing possible is limited by the reliability required by the agent. High levels of sharing make for efficient use of the resources but run the risk of failure

due to contention for the resource among the agents. Upper level agents 120 can indicate or stipulate the reliability that they require. The higher layer services 126 can set their sharing parameters to allocate adequate resources so that the chance of failure due to resource contention among agents is set at an acceptable level. That is, more resources or capability of a resource can be given to applications requiring more reliability to minimize the chances of contention.

Capacity of a resource may be measured using conventional means and in a preferred embodiment measured and determined using a service model. Service models are well known in the art and are successfully used for modeling, simulation and resource determination according to probability distributions such as Poisson, Erlang A., Band C. or Engset. The service model can be used to translate a request by a bidding agent for a specific quantity and quality of service from an upper level into a specific resource capacity which can be shared among agents. This leads to the ability of the system to incrementally allocate services. If the higher layer receives a request for a specific service level, it will match this through service models with the capacity and quality of service which must be obtained from lower level services to meet this requirement. This is achieved through the mechanism of 'resource sets'. The resource set is a detailed description of the resources available and may be characterized as a set of descriptors. Each of the descriptors describes a set of resources which can together be used to satisfy a service request to a specified level of service along with the capacity of each resource required. The resource set can be implemented as a set of tuples or key value pairs loaded or configured in the system.

For example; a resource set could be:

Resource Descriptor (1),

Resource (name of resource), Capacity (capacity of resource)

Resource (name of resource), Capacity (capacity of resource)

•

•

•

Resource Descriptor (2),

Resource (name of resource), Capacity (capacity of resource)

Resource (name of resource), Capacity (capacity of resource)

-
-
-

-
-
-

A service level may be met with one of several configurations of lower level agents. Each of these configurations can be described as a resource set. Within the higher agent, each service level is matched with a group of resource sets which can accomplish it. The higher agent can combine the services of one or more lower level agents to create its required service. It estimates the capacity required of each through service models which translate its service level request into an appropriate service level at the lower agent levels. If it has sufficient quantity of one of the lower level services, it can grant the request immediately. If not it must request the capacity.

As shown in the diagram, the lower level agent 122 is sharing its services among several higher level agents 120. To do this sharing fairly, it must have guidance from the system as to the relative priorities it should put on requests from different agents. Although this may be done simply but crudely by the endowment of priority, in the preferred embodiment this is done through a mechanism called 'sliding constraints' based on the concept of 'importance'. This 'importance' measure has been endowed or encoded by the quantity of economic utility which has been supplied to each of the higher level agents. It will make available to each of these upper level agents 120 capacity within a constraint based on this importance. This can be done incrementally. During operation the upper level agents 120 will have been assigned capacities to meet their requests. With a new request, the lower level agent 122 can meet this request through use of a proportion of its unallocated capacity and portions of the capacity of less important agents. This means that if necessary the lower agents can take capacity away from less important agents to meet the needs of the more important. Thus the capacity supplied to higher level agents are constraints on their usage of the lower level service which slide with requests and their relative importance.

Upon a service request, the lower level agent will reassess the service it supplies to its higher layers and make sufficient reallocations to meet it. It will then inform the affected higher layers of the reallocations. These will in turn reallocate their capacity among the agents that they are serving.

This system has the benefit of being adaptive and self-configuring. Resources and services will be allocated on the basis of the needs of the overall system as indicated by the relative importance factors. These factors can be changed during operation to meet new contingencies in service opportunities or to address failure. The resource and service allocation of the system will automatically adjust to meet these new requirements. For example if a resource assigned to an important application fails. The necessary amount of resource can be removed from the capacity assigned to another agent or agents. The agent or agents can then handle this as it or they would handle a service request and reassign its resource to upper level resources. The important agent would be maintained at the expense of those at lesser importance. If a service suddenly becomes more important, for example to handle an emergency it can be given a higher importance. As the newly important service is exercised, it can take resources from other agents. Thus the mechanism is adaptive to the needs of the system by being able to reallocate on request and it is self-configuring since it handles the contention for resources automatically.

The exponent can be changed to balance the amount of resources that will be taken away from low priority resources. An exponent close to 1 will cause resources to be taken preferentially from low priority agents. Exponents closer to 0 will cause resources to be removed from all lower bidding agents. This exponent may be tailored to the agents needs. The trade off can be done considering the factor it is better to deny a service request or to slow the response to all service request by accommodating it. This trade off can be done in the context of the shared resource and can be tuned to its needs.

Having thus described the resource sharing system which embodies the present invention, a description is provided with reference to Figures 15 and 16 of the inventive sliding constraints algorithm.

Figure 15 defines the parameters used to determine the dynamic priorities of request classes for the case of two request classes. For each request class a setting specifies the minimum allocation to that class, $\mathbf{min}_i$. This is the portion of the resource deterministically allocated to the request class. Obviously, the sum of all minimum allocations to request classes cannot exceed 100% of the resource.

The rest of the resource is allocated statistically. In statistical allocation, a request can be allocated the resource if no other higher priority request has been scheduled at the time the resource becomes available. The extent of statistical resource allocation to a request class is specified by its maximum allocation, $\mathbf{max}_i$. More specifically, the range between $\mathbf{min}_i$ and $\mathbf{max}_i$, is allocated statistically.

Statistical sharing is advantageous over deterministic sharing in that surplus resources that would otherwise be idle can be reassigned to lower-priority request classes. A disadvantage of statistical sharing is the well-known problem of overbooking. The present invention provides a solution to the problem of overbooking through dynamic priority adjustment for the statistically shared portion of the resource.

Thus, according to the present invention, resource requests are satisfied in an interleaving manner, whereby each increase in average resource allocation $\mathbf{u}_i$, results in a decrease in the priority $\mathbf{p}_i$ of the corresponding request class. Upon decreasing sufficiently, another request class will obtain higher priority so that its resource requests may begin to be satisfied until its priority has decreased to below the priority of yet another request class etc.

It should be noted that wherever this disclosure refers to a single resource, a resource with multiple units or multiple resources may be used, without restriction on the methodology of the invention. A multiple resource is typically modeled as a discrete number of resource units, for example, a trunk group with 24 discrete trunk. In the case of multiple resources, multiple resource requests can be satisfied concurrently as long as resource units remain available.

The priority of a request class is adjusted proportionally to the moving average allocation. If the moving average allocated to a request class is $\mathbf{u}_i$, then the priority can be determined by the following formula:

$$\mathbf{p}_i = 1.0 \qquad \text{if } \mathbf{u}_i <= \mathbf{min}_i$$

$$\mathbf{p}_i = 1.0 - (\mathbf{u}_i - \mathbf{min}_i)/(\mathbf{max}_i - \mathbf{min}_i) \qquad \text{if } \mathbf{min}_i < \mathbf{u}_i <= \mathbf{max}_i$$

$$\mathbf{p}_i = 0.0 \qquad \text{if } \mathbf{u}_i >= \mathbf{max}_i$$

As indicated above, the priority $\mathbf{p}_i$ "slides" with the average resource allocation $\mathbf{u}_i$, and the priority imposes a constraint on when the requests of a request class can be satisfied: namely, when there are no other requests from higher-priority request classes waiting for the resource.

Figure 16 illustrates the steps in processing a resource requests in accordance with the present invention. Requests arrive in queues 1A, 1B, 1C, etc., with one queue per request class, and are queued in first-come first-served (FCFS) order. If the resource is currently allocated, a pending request must wait for the resource to be relinquished by the application that currently holds it.

The sliding constraints module 3 dynamically assigns a priority to each queue 1A, 1B, 1C, etc. based on the moving averages of their request classes. The request at the head of the queue with the highest priority gets selected. If priorities are equal, the system allocates the resource to the request that has been queued the longest (allocate module 5).

On allocating the resource, the sliding constraints module 3 is informed to update its moving averages. It should be noted that this update step is only required in the case of a multiple resource, because requests can be allocated while other applications have not yet relinquished the resources allocated to them.

Sometime later in the execution thread of the application using the resource, the resource is relinquished (relinquish module 7), in response to which the sliding constraints module 3 is informed to update its moving averages.

In order to provide additional implementation details of the sliding constraints module 3, allocate module 5 and relinquish module 7, pseudocode is set forth herein below which makes use of the following definitions, expressed as Java code.

First, a Resource class is defined that maintains the number of resource units available, the specification of request classes, and queues for waiting requests. It provides mainly a framework for the remaining definition. In practice, a system using the sliding constraints mechanism could manage this information in a different manner, for example, if it needs to manage multiple resources and represent separate information about each.

```
// Assuming that we have access to a Queue class that can be found in
commonly available class libraries, we define resources in the class
Resource. Each resource manages the number of units available, request
classes, and queues for waiting requests.

class Resource {
  static long units;                        // number of units
  static RequestClass requestClass[];       // requestClass[i] holds the
operating parameters for request class i
  static Queue waitingRequests[];           // waitingRequests[i] holds the
requests for request class i
}
```

Next, a RequestClass is defined to store the parameters associated with a request class such as the average allocation, minimum and maximum allocation. It also maintains the information needed to compute the moving average. There are several known mechanisms for computing moving averages (simple, exponential, triangular, etc.). In the implementation described herein, a time window is used over which a simple, unweighted average is computed. The time window is represented by an array, the window length, and indices to advance and access the window. For practical reasons, in this example the resource has been divided into discrete units, so that the algorithm can be implemented in a microcontroller. However, there is no inherent restriction on the unit size. Specifically, it is contemplated that only one be allocated at a time.

The methods of the RequestClass allow the modules to update the moving average, notify the RequestClass that a resource has been allocated or relinquished, and to compute the priority for the request class. In the present example, the update function is invoked once with every clock tick, or a reasonable multiple thereof, which sets the basic interval within which a

resource request can be satisfied. All resource requests received within the interval are considered to have arrived simultaneously.

```
// Assume that we have defined a class RequestClass to store the parameters
associated with a request class.

class RequestClass {
  double u;                    // average allocation
  double min;                  // minimum allocation
  double max;                  // maximum allocation
  int w;                       // length of the time window for computing
the moving average
  long unitsHeld[];            // w most recent allocations
  int i,j,k;                   // window indices

  // Compute the moving average. We expect this method to be invoked at
every clock tick to recompute the current moving average
  public double update() {
    k = i; i = (i+1)%w;        // advance window index i by 1, store
previous index in k
    if (j<w) j++;              // advance window size j (just until we fill
the window, subsequently j equals w)
    long units = 0;            // sum units held over the time window
    for (l=0; l<=j; l++) units += unitsHeld[l];
                               // compute new moving average
    u = units/(w*Resource.units);
  }

  // Update the currently held units on allocating
  public resourceAllocated(Request r) {
    unitsHeld[i] = unitsHeld[k] + r.units;
  }

  // Update the currently held units on relinquishing
  public resourceRelinquished(Request r) {
    unitsHeld[i] = unitsHeld[k] - r.units;
  }

  // Compute the priority for this resource class based on its current
moving average
  public double priority() {
    if (u <= min) return 1.0;
    if (u <= max) return 1.0 - (u-min)/(u-max);
    return 0.0;
  }
}
```

Finally, a Request class is defined to hold the parameters of a request. It contains the number of units requested and a timestamp used to resolve conflicts if two requests should have the same the same priority.

```
// Above we assumed the existence of a Request class to hold the parameters
of a request.

class Request {
  long units;                  // units requested
  long timestamp;              // time when request was placed (as measured
from the system start, initially 0)
}
```

Based on the foregoing definitions, exemplary pseudocode of the three modules is set forth below. For reasons of conciseness, the Java code is not provided, but rather an indication of which Java method defined above needs to be invoked from within the code. The first module is the sliding constraints module 3.

At each clock tick (or reasonable multiple thereof):

Update the moving averages for each request class defined in the Resource class using the RequestClass.update() method.

Sort the request classes by their priority (computed using the RequestClass.priority() method), including only request classes that have waiting requests.

If there more than one request class happen to have the same priority, compare the timestamps of the requests at the head of each request queue.

Pass the selected request (if there was one) to the allocate module 5.

The next module is the pseudocode for the allocate module 5:

Inform the request class of the request that the request has been allocated, passing the request as a parameter to the RequestClass.resourceAllocated() method. This updates the information held by the resource class as to its current allocation.

The application now uses the resource (outside the scope of the mechanism). When it relinquishes the resource, the pseudocode for the relinquish module 7 is executed.

Inform the request class of the request that the request has been relinquished, passing the request as a parameter to the RequestClass.ResourceRelinquished() method. This updates the information held by the resource class as to its current allocation.

The following example of a "call center" of a small development shop is an illustration of a practical technical application of the present invention. In the small development shop of the present example software developers are required to double up as helpdesk agents. The sliding constraints mechanism may be used to implement a policy

according to which a software developer only receives up to a certain quota of all helpdesk calls. The minimum and maximum resource allocations for the two roles that the software developer can assume permit detailed specification of how much of the developer's time should be spent on programming (SW) and how much on helpdesk calls (HD), as compared to simply specifying static priorities.

For each role, the minimum and maximum amounts of time the developer should spend exclusively in that role can be specified, as well as how much of his time can float between the two activities. For example, software developer Bob can have the following policy settings established for him:

SW role:	$\mathbf{min}_{SW} = 0.50$	$\mathbf{max}_{SW} = 1.00$
HD role:	$\mathbf{min}_{HD} = 0.20$	$\mathbf{max}_{HD} = 0.50$

If the software development task can be broken down into chunks of, say, 15 min each after which Bob can be interrupted to take a helpdesk call, and that answering a helpdesk call also takes 15 min. A 7.5h workday can then be divided into 30 chunks. The settings above translate into chunks as follows:

15 chunks reserved for the SW role
6 chunks blocked off for the HD role
9 chunks float allocated on demand

If, for example, during the first 15 chunks Bob does not receive a single helpdesk call (unlikely as this may be), the priority for helpdesk requests will be 1.0 for at least the next 6 calls. For purposes of illustration only, for a single work day in Bob's life, the moving averages may be computed over a time window of 30 chunks, as shown in Table A, below.

Requests arrive during each time unit or chunk. For the first 15 chunks, only SW requests arrive. Subsequently, SW and HD requests arrive simultaneously. The request column of Table A shows the request with the highest priority after the previous time step,

but before the request is accounted for in the moving average. So for example, row 16 reads as:

- At chunk 15 the SW role has priority 0.0 and the HD role 1.0.
- Thus the HD request is selected and Bob receives a helpdesk call.
- On answering the call, the moving averages are updated.
- The new priorities are 0.12 for the SW role and 1.00 for the HD role.

It will be noted in Table A, below, that the requests typically interleave. For example at chunk 25, the SW role is selected because its priority after the previous resource allocation (chunk 24) was 0.64 which is higher than the priority for the HD role, 0.60.

TABLE A

t	Request	$\#_{SW}$	u_{SW}	p_{SW}	$\#_{HD}$	u_{HD}	p_{HD}
0	SW	1	1.00	0.00	0	0.00	1.00
1	SW	2	1.00	0.00	0	0.00	1.00
	...						
15	SW	16	1.00	0.00	0	0.00	1.00
16	HD	16	0.94	0.12	1	0.06	1.00
17	SW	17	0.94	0.11	1	0.06	1.00
18	HD	17	0.89	0.21	2	0.11	1.00
19	HD	17	0.85	0.30	3	0.15	1.00
20	HD	17	0.81	0.38	4	0.19	1.00
21	HD	17	0.77	0.45	5	0.23	0.91
22	HD	17	0.74	0.52	6	0.26	0.80
23	HD	17	0.71	0.58	7	0.29	0.69
24	HD	17	0.68	0.64	8	0.32	0.60
25	SW	18	0.69	0.62	8	0.31	0.64
26	HD	18	0.67	0.67	9	0.33	0.56
27	SW	19	0.68	0.64	9	0.32	0.60
28	HD	19	0.66	0.69	10	0.34	0.52
29	SW	20	0.67	0.67	10	0.33	0.56

Various alternatives and embodiments of the invention are possible. Instead of a continuous sliding function for the priorities, the use of monotonously decreasing discrete step functions is also contemplated, with resulting ease of digital implementation of the inventive algorithm in a microcontroller.

In the given formulation of the mechanism set forth above, the priority assigned to a request is the priority *before* its pending impact on the moving average is accrued for.

However, if the sliding constraints mechanism is to be used to allocate several units of a multiple resource at once (for example, multiple blocks of computer memory), a mechanism must be provided to ensure that there will be enough units available. This can be accomplished by either:

- Setting the priority to that which would be obtained *after* allocating the requested number of resource units.
- Allocating the resource units *one by one* whenever that is possible (for example, for network packets) interleaved with other simultaneous requests.

Another variation on the embodiment wherein multiple resource units are requested at once, is contemplated in situations where the total number of units requested exceeds the number of available units. In this scenario, the sliding constraints mechanism of the present invention can be used to compute permissible request sizes for each request class. Permissible sizes are computed as geometrically proportional to their priorities.

For example, if there are 3 units of a given resource (for example, 3 trunks), but the more simultaneous requests than units available:

3 units in a class 1 request at priority 0.5
3 units in a class 2 request at priority 1.0

Then the 3 resource units can be divided up in geometric proportions of the priorities as follows:

(priority of class 1) : (priority of class 2) = 1 : 2

Assigned to the class 1 request (its permissible size): 1 units
Assigned to the class 2 request (its permissible size): 2 units

For a total of: 3 units

Other variations and alternatives are possible without departing from the sphere and scope of the invention as set forth in the claims appended hereto.